# Low Density Parity Check Code for Rate 7/8

September 2005

# CONTENTS


| Section | | Page |
|---|---|---|
| 1. | Background | 1 |
| 2. | Introduction | 2 |
| 3. | Baselined (8176,7156) LDPC Code | 3 |
| 4. | Encoding | 5 |
| 5. | Recommended Shortened (8160, 7136) Code | 6 |
| 6. | Randomization and Synchronization | 8 |
| 7. | References | 8 |
| 8. | Appendix A – Generator Matrix Circulant table | 9 |
| 9. | Appendix B – Complexity | 11 |
| 10. | Appendix C – FPGA test results | 12 |

# LIST OF FIGURES

**Section** **Page**

Figure 1. Example of a 15 x 15 circulant matrix ........ 2
Figure 2. Example of a quasi-cyclic matrix ........ 3
Figure 3. Base Parity Check Matrix of the (8176, 7156) LDPC code ........ 3
Figure 4. Scatter Chart of Parity Check Matrix ........ 4
Figure 5. Systematic Circulant Generator Matrix ........ 6
Figure 6. Shortened Codeword ........ 7
Figure 7. Bit Error Rate Test Results ........ 12
Figure 8. Block Error Rate Test Results ........ 13

# LIST OF TABLES

**Section** **Page**

Table 1. Specification of Circulants ..... 4
Table 2. Table of Circulants for the Generator Matrix ..... 9

# 1. BACKGROUND

The mid-1990's were highlighted by the rediscovery of Low Density Parity Check codes (LDPCC) in the field of channel coding [1]. Originally invented by R. Gallager in his PhD thesis in 1961 [2], this coding technique was largely forgotten for more than 30 years. The primary advance in LDPCC is the discovery of an iterative decoding algorithm, now called Belief Propagation (BP) decoding, which offers near-optimum performance for large linear LDPCC at a manageable complexity. LDPCC performance gains were difficult to technologically realize in the early 1960's. Several decades of VLSI development has finally made the implementation of these codes practical.

The original construction, now called Gallager LDPCC, has come to be regarded as a special class of LDPCC. Recent advances in LDPC code construction have resulted in the development of new codes with (arguably) improved performance over Gallager LDPCC. One class of these codes, irregular LDPCC [3], demonstrates improved performance in the waterfall region. Disadvantages of irregular codes, however, include an increase, in general, in the number of iterations required for decoding convergence and an unequal error protection between code bits resulting from the irregular structure. Another class of LDPCC developed using algebraic construction based on finite geometries [4] has shown to provide very low error floors and very fast iterative convergence. These qualities make these codes a good fit for near Earth applications where very high data rates and high reliability are the driving requirements.

## 2. INTRODUCTION

A linear block code is designated in this orange book by (n, k) where n is the length of the codeword (or block) and k is the length of the information sequence. LDPC codes are linear block codes in which the ratio of the total number of 1's to the total number of elements in the parity check matrix is $\ll 0.5$. The distribution of the 1's determine the structure and performance of the decoder. An LDPC code is defined by its parity check matrix. The k x n generator matrix which is used to encode a linear block code can be derived from the parity check matrix through linear operations. (The reader is encouraged to review [8] for an overview of linear block codes).

The LDPC code considered in this specification is a member of a class of codes called Quasi-Cyclic codes. The construction of these codes involves juxtaposing smaller circulants (or cyclic submatrices) to form a larger parity check or base matrix.

An example of a circulant is shown in Figure 1. Notice that every row is one bit right cyclic shift (where the end bit is wrapped around to the beginning bit) of the previous row. The entire circulant is uniquely determined and specified by its first row. For this example the first row has 4 1's or a row weight of 4.

$$\begin{bmatrix}
0&0&0&0&0&0&0&1&1&0&1&0&0&0&1\\
1&0&0&0&0&0&0&0&1&1&0&1&0&0&0\\
0&1&0&0&0&0&0&0&0&1&1&0&1&0&0\\
0&0&1&0&0&0&0&0&0&0&1&1&0&1&0\\
0&0&0&1&0&0&0&0&0&0&0&1&1&0&1\\
1&0&0&0&1&0&0&0&0&0&0&0&1&1&0\\
0&1&0&0&0&1&0&0&0&0&0&0&0&1&1\\
1&0&1&0&0&0&1&0&0&0&0&0&0&0&1\\
1&1&0&1&0&0&0&1&0&0&0&0&0&0&0\\
0&1&1&0&1&0&0&0&1&0&0&0&0&0&0\\
0&0&1&1&0&1&0&0&0&1&0&0&0&0&0\\
0&0&0&1&1&0&1&0&0&0&1&0&0&0&0\\
0&0&0&0&1&1&0&1&0&0&0&1&0&0&0\\
0&0&0&0&0&1&1&0&1&0&0&0&1&0&0\\
0&0&0&0&0&0&1&1&0&1&0&0&0&1&0
\end{bmatrix}$$

**Figure 1. Example of a 15 x 15 circulant matrix**

An example of a quasi-cyclic parity check matrix is shown in Figure 2. In this case, a quasi-cyclic 10 x 25 matrix is formed by an array of 2 x 5 circulant submatrices of size 5 x 5. To unambiguously describe this matrix, only the position of the 1's in the first row of every circulant submatrix and the location of each submatrix within the base matrix is needed.

$$
\left[\begin{array}{ccccc|ccccc|ccccc|ccccc|ccccc}
1&0&0&1&0&1&1&0&0&0&0&1&0&0&1&0&1&1&0&0&1&0&0&0&1\\
0&1&0&0&1&0&1&1&0&0&1&0&1&0&0&0&0&1&1&0&1&1&0&0&0\\
1&0&1&0&0&0&0&1&1&0&0&1&0&1&0&0&0&0&1&1&0&1&1&0&0\\
0&1&0&1&0&0&0&0&1&1&0&0&1&0&1&1&0&0&0&1&0&0&1&1&0\\
0&0&1&0&1&1&0&0&0&1&1&0&0&1&0&1&1&0&0&0&0&0&0&1&1\\
\hline
0&1&0&0&1&0&0&0&1&1&0&0&1&0&1&1&0&1&0&0&1&1&0&0&0\\
1&0&1&0&0&1&0&0&0&1&1&0&0&1&0&0&1&0&1&0&0&1&1&0&0\\
0&1&0&1&0&1&1&0&0&0&0&1&0&0&1&0&0&1&0&1&0&0&1&1&0\\
0&0&1&0&1&0&1&1&0&0&1&0&1&0&0&1&0&0&1&0&0&0&0&1&1\\
1&0&0&1&0&0&0&1&1&0&0&1&0&1&0&0&1&0&0&1&1&0&0&0&1
\end{array}\right]
$$

**Figure 2. Example of a quasi-cyclic matrix**

Constructing parity check matrices in this manner produces two positive features:

1. the encoding complexity can be made linear with the code length or parity bits using shift registers, and
2. encoder and decoder routing complexity in the interconnections of integrated circuits is reduced.

# 3. BASELINED (8176,7156) LDPC CODE

The parity check matrix for the (8176, 7156) LDPC code is formed by using a 2 x 16 array of 511 x 511 square circulants. This creates a parity check matrix of dimension 1022 x 8176. The structure of the parity check base matrix is shown in Figure 3.

$$
\begin{bmatrix}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} & A_{1,5} & A_{1,6} & A_{1,7} & A_{1,8} & A_{1,9} & A_{1,10} & A_{1,11} & A_{1,12} & A_{1,13} & A_{1,14} & A_{1,15} & A_{1,16} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} & A_{2,5} & A_{2,6} & A_{2,7} & A_{2,8} & A_{2,9} & A_{2,10} & A_{2,11} & A_{2,12} & A_{2,13} & A_{2,14} & A_{2,15} & A_{2,16}
\end{bmatrix}
$$

**Figure 3. Base Parity Check Matrix of the (8176, 7156) LDPC code**

Each $A_{i,j}$ is a 511 x 511 circulant. The row weight of the each of the 32 circulants is 2, i.e. there are two 1's in each row. The total row weight of each row in the parity check matrix is 2 x 16 or 32. The column weight of each circulant is also 2, i.e. there are two 1's in each column. The total weight of each column in the parity check matrix is 2 x 2 or 4. The position of the 1's in each circulant is defined in table 1. A scatter chart of the parity check matrix is shown in figure 4 where every 1 bit in the matrix is represented by a point.

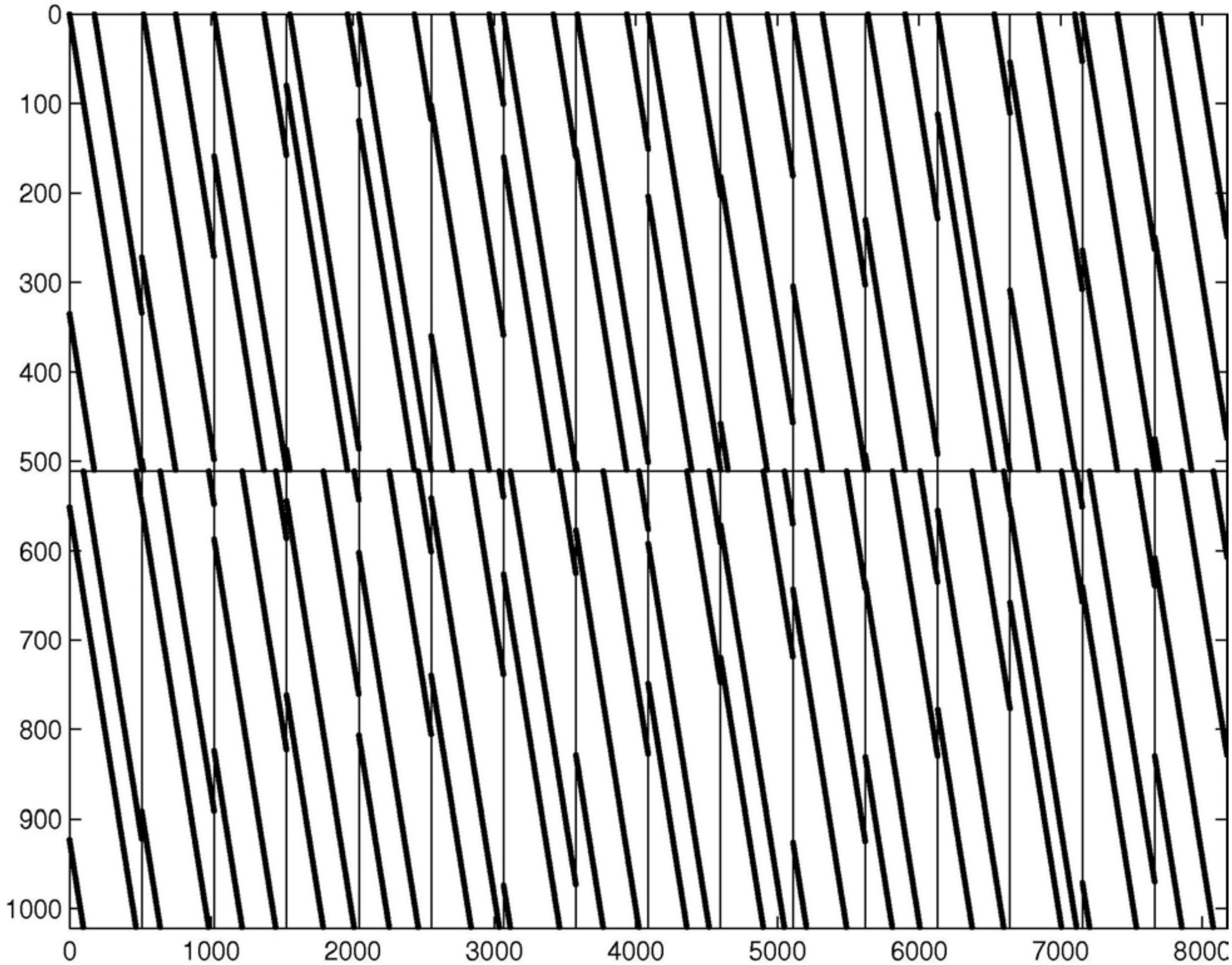


**Figure 4. Scatter Chart of Parity Check Matrix**

**Table 1. Specification of Circulants**

| Circulant | 1's position in 1st row of circulant | Absolute 1's position in 1st row of Parity Check Matrix |
|---|---|---|
| $A_{1,1}$ | 0, 176 | 0, 176 |
| $A_{1,2}$ | 12, 239 | 523, 750 |
| $A_{1,3}$ | 0, 352 | 1022, 1374 |
| $A_{1,4}$ | 24, 431 | 1557, 1964 |
| $A_{1,5}$ | 0, 392 | 2044, 2436 |
| $A_{1,6}$ | 151, 409 | 2706, 2964 |
| $A_{1,7}$ | 0, 351 | 3066, 3417 |
| $A_{1,8}$ | 9, 359 | 3586, 3936 |
| $A_{1,9}$ | 0, 307 | 4088, 4395 |
| $A_{1,10}$ | 53, 329 | 4652, 4928 |
| $A_{1,11}$ | 0, 207 | 5110, 5317 |

| | | |
|---|---|---|
| $A_{1,12}$ | 18, 281 | 5639, 5902 |
| $A_{1,13}$ | 0, 399 | 6132, 6531 |
| $A_{1,14}$ | 202, 457 | 6845, 7100 |
| $A_{1,15}$ | 0, 247 | 7154, 7401 |
| $A_{1,16}$ | 36, 261 | 7701, 7926 |
| $A_{2,1}$ | 99, 471 | 99, 471 |
| $A_{2,2}$ | 130, 473 | 641, 984 |
| $A_{2,3}$ | 198, 435 | 1220, 1457 |
| $A_{2,4}$ | 260, 478 | 1793, 2011 |
| $A_{2,5}$ | 215, 420 | 2259, 2464 |
| $A_{2,6}$ | 282, 481 | 2837, 3036 |
| $A_{2,7}$ | 48, 396 | 3114, 3462 |
| $A_{2,8}$ | 193, 445 | 3770, 4022 |
| $A_{2,9}$ | 273, 430 | 4361, 4518 |
| $A_{2,10}$ | 302, 451 | 4901, 5050 |
| $A_{2,11}$ | 96, 379 | 5206, 5489 |
| $A_{2,12}$ | 191, 386 | 5812, 6007 |
| $A_{2,13}$ | 244, 467 | 6376, 6599 |
| $A_{2,14}$ | 364, 470 | 7007, 7113 |
| $A_{2,15}$ | 51, 382 | 7205, 7536 |
| $A_{2,16}$ | 192, 414 | 7857, 8079 |

Note that the numbers in the second column represent the relative column position of the 1's in the first row of each circulant. Since there are only 511 possible positions, these numbers can only range from 0 to 510. The third column represents the absolute position of the 1's in the parity-check matrix. There are exactly 8176 possible; therefore these numbers can only range from 0 to 8175.

# 4. ENCODING

The encoder can be designed using the method given in [6]. The generator matrix of the (8176, 7156) code consists of two parts. The first part is a 7154 x 8176 submatix in systematic-circulant form as shown in Figure 5. It consists of a 7154 x 7154 identity matrix and two columns of 511 x 511 circulants $B_{i,j}$'s, each column consisting of 14 circulants. The I's are the 511 x 511 identity submatrices and the 0's are the all zero 511 x 511 submatrices. The second part consists of two independent rows. The first part generates a (8176, 7154) LDPC subcode of the (8176, 7156) code. Each codeword in the subcode consists of 7154 information bits and 1022 parity-check bits. For reason given in Section 4.1, there are advantages in using the subcode implementation. The circulants $B_{i,j}$'s are constructed based on the algorithm given below:

1. From Figure 3 and Table 1, define $D = \begin{bmatrix} A_{1,15} & A_{1,16} \\ A_{2,15} & A_{2,16} \end{bmatrix}$, which is a 1022 x 1022 matrix.
2. Let $u = (1\ 0\ 0\ 0\ \ldots.\ 0)$ be the unit 511 tuple, i.e. a vector quantity of length 511 with a "1" at the leftmost position and "0"s in the rest.
3. Define $z_i = (b_{i,1} \quad b_{i,2})$ where $i$ = 1, 2, ..., 14 and the $b_{i,j}$'s are first row of the $B_{i,j}$ circulants

4. Define $\mathrm{M}_i = \begin{bmatrix} \mathrm{A}_{1,i} \\ \mathrm{A}_{2,i} \end{bmatrix}$, where $i$ = 1, 2, …, 14. (Note that the parity check matrix can now be represented as: $[\mathrm{M}_1\ \mathrm{M}_2 \ldots \mathrm{M}_{14}\ \mathrm{D}]$)
5. Since the rank of D is 1020 not 1022, there are two linearly dependent columns, 511th and 1022nd. Set the 511th and 1022nd elements of $z_i$ to zero and solve $\mathrm{M}_i u^{\mathrm{T}} + \mathrm{D} z_i^{\mathrm{T}} = 0$ for $z_i$, where $i$ = 1, 2, …, 14 and T superscript represents matrix transpose.
6. The $b_{i,j}$'s can be extracted from the $z_i$'s. (They are numerically tabulated in Appendix A.)

There are many ways to design the encoder based on the generator matrix in Figure 5. These schemes have complexities that are proportional to the length of the codeword or parity check bits [6].

$$
\begin{bmatrix}
I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{1,1} & B_{1,2} \\
0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{2,1} & B_{2,2} \\
0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{3,1} & B_{3,2} \\
0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{4,1} & B_{4,2} \\
0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{5,1} & B_{5,2} \\
0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{6,1} & B_{6,2} \\
0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & 0 & B_{7,1} & B_{7,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & 0 & B_{8,1} & B_{8,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & 0 & B_{9,1} & B_{9,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & 0 & B_{10,1} & B_{10,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & 0 & B_{11,1} & B_{11,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & 0 & B_{12,1} & B_{12,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & 0 & B_{13,1} & B_{13,2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & I & B_{14,1} & B_{14,2}
\end{bmatrix}
$$

**Figure 5. Systematic Circulant Generator Matrix**

# 5. RECOMMENDED SHORTENED (8160, 7136) CODE

Using the generator matrix given by Figure 5, an encoder can be implemented using a circuit described in [6]. This encoder generates a (8176, 7154) LDPC subcode of the (8176, 7156) code. Current spacecraft and ground systems manipulate and process data at 32-bit computer word size. Neither (8176, 7154) or (8176, 7156) is a multiple of 32. It is beneficial to shorten the codeword to the dimensions of (8160, 7136). In other words, by shortening the information sequence to 7136 through the use of 18 bits of virtual fill, the (8176, 7154) subcode encoder can be used. This is accomplished by encoding the virtual fill bits with zeros but not transmitting them; thus the total codeword length becomes 8158. Note that it is not necessary to add two independent rows to the generator matrix to encode the full (8176, 7156) code because these bits would be shortened anyway and so the subcode is sufficient and less complicated for this application.  Since the codelength of 8158 is two bits shy of 8160, an exact multiple of 32, two bits of actual transmitted zero fill are appended to end of the codeword to achieve a shortened code dimension of (8160, 7136) bits or (1020, 892) octets or (255, 223) 32-bit words.  The shortened codeword is shown in Figure 6.

The received shortened codeword would require the removal of the 2 zero fill bits prior to decoding.  The decoder would then reproduce the 18 virtual fill zeros after processing but would, in general, not pass these 18 zeros on to the ground equipment.

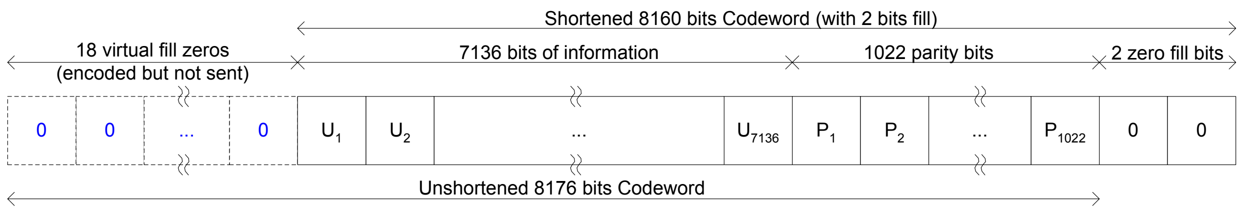


**Figure 6.  Shortened Codeword**

# 6. RANDOMIZATION AND SYNCHRONIZATION

The use of the recommended shortened (8160, 7136) LDPC code does not guarantee sufficient bit (symbol) transitions to acquire or maintain bit (symbol) synchronization. It is highly recommended that a pseudo-randomizer be used after encoding in accordance to CCSDS recommendation 131.0-B-1, TM Synchronization and Channel Coding. Blue Book. Issue 1. September 2003 Section 7.

In addition, frame (codeword) synchronization is required so that the receiver can identify the beginning of the frame (codeword) for proper decoding. The use of an attached sync marker (ASM) as specified in CCSDS recommendation 131.0-B-1, TM Synchronization and Channel Coding. Blue Book. Issue 1. September 2003 Section 6.6 is required. Note that the ASM is not pseudo-randomized.

# 7. REFERENCES

[1] D. J. C. MacKay and R. M. Neal, "Near Shannon limit performance of low density parity check codes," *Electro. Lett.*, vol. 32, pp. 1645-1646, Aug. 1996.

[2] R. G. Gallager, "Low density parity check codes," *IRE Trans. Inform. Theory*, IT-8, pp. 21-28, Jan. 1962.

[3] T. Richardson and R. Urbanke, "Design of capacity-approaching low density parity check codes," *IEEE Trans. Inform. Theory*, vol. 47, pp. 619-637, Feb. 2001.

[4] Y. Kou, S. Lin, and M. P. C. Fossorier, "Low-density parity-check codes based on finite geometries: a rediscovery and new results," *IEEE Trans. Information Theory*, vol. 47, pp. 2711-2736, Nov. 2001.

[5] W. Fong, "White Paper for Low Density Parity Check (LDPC) Codes for CCSDS Channel Coding Blue Book," CCSDS P1B Channel Coding Meeting: Houston, TX, Oct. 2002.

[6] Z. Li, L. Chen, L. Zeng, S. Lin, and W. Fong, "Efficient Encoding of Quasi-Cyclic Low Density Parity Check Codes," accepted for publication in: *IEEE Transactions on Communication*.

[7] J. Heo, "Analysis of scaling soft information on low density parity check code," *Electro. Lett.*, vol. 39, pp. 219-221, Jan. 2003.

[8] S. Lin and D. Costello, Jr. *Error Control Coding, 2nd Ed.* New Jersey: Pearson Prentice Hall, 2004.

# 8. APPENDIX A – GENERATOR MATRIX CIRCULANT TABLE

**Table 2. Table of Circulants for the Generator Matrix**

| Circulant | 1st row of circulant |
|---|---|
| $B_{1,1}$ | 55BF56CC55283DFEEFEA8C8CFF04E1EBD9067710988E25048D67525426939E2068D2<br>DC6FCD2F822BEB6BD96C8A76F4932AAE9BC53AD20A2A9C86BB461E43759C |
| $B_{1,2}$ | 6855AE08698A50AA3051768793DC238544AF3FE987391021AAF6383A6503409C3CE9<br>71A80B3ECE12363EE809A01D91204F1811123EAB867D3E40E8C652585D28 |
| $B_{2,1}$ | 62B21CF0AEE0649FA67B7D0EA6551C1CD194CA77501E0FCF8C85867B9CF679C18B<br>CF7939E10F8550661848A4E0A9E9EDB7DAB9EDABA18C168C8E28AACDDEAB1E |
| $B_{2,2}$ | 64B71F486AD57125660C4512247B229F0017BA649C6C11148FB00B70808286F1A9790<br>748D296A593FA4FD2C6D7AAF7750F0C71B31AEE5B400C7F5D73AAF00710 |
| $B_{3,1}$ | 681A8E51420BD8294ECE13E491D618083FFBBA830DB5FAF330209877D801F92B5E0<br>7117C57E75F6F0D873B3E520F21EAFD78C1612C6228111A369D5790F5929A |
| $B_{3,2}$ | 04DF1DD77F1C20C1FB570D7DD7A1219EAECEA4B2877282651B0FFE713DF338A632<br>63BC0E324A87E2DC1AD64C9F10AAA585ED6905946EE167A73CF04AD2AF9218 |
| $B_{4,1}$ | 35951FEE6F20C902296C9488003345E6C5526C5519230454C556B8A04FC0DC642D682<br>D94B4594B5197037DF15B5817B26F16D0A3302C09383412822F6D2B234E |
| $B_{4,2}$ | 7681CF7F278380E28F1262B22F40BF3405BFB92311A8A34D084C086464777431DBFD<br>DD2E82A2E6742BAD6533B51B2BDEE0377E9F6E63DCA0B0F1DF97E73D5CD8 |
| $B_{5,1}$ | 188157AE41830744BAE0ADA6295E08B79A44081E111F69BBE7831D07BEEBF76232E<br>065F752D4F218D39B6C5BF20AE5B8FF172A7F1F680E6BF5AAC3C4343736C2 |
| $B_{5,2}$ | 5D80A6007C175B5C0DD88A442440E2C29C6A136BBCE0D95A58A83B48CA0E7474E<br>9476C92E33D164BFF943A61CE1031DFF441B0B175209B498394F4794644392E |
| $B_{6,1}$ | 60CD1F1C282A1612657E8C7C1420332CA245C0756F78744C807966C3E1326438878B<br>D2CCC83388415A612705AB192B3512EEF0D95248F7B73E5B0F412BF76DB4 |
| $B_{6,2}$ | 434B697B98C9F3E48502C8DBD891D0A0386996146DEBEF11D4B833033E05EDC28F8<br>08F25E8F314135E6675B7608B66F7FF3392308242930025DDC4BB65CD7B6E |
| $B_{7,1}$ | 766855125CFDC804DAF8DBE3660E8686420230ED4E049DF11D82E357C54FE256EA0<br>1F5681D95544C7A1E32B7C30A8E6CF5D0869E754FFDE6AEFA6D7BE8F1B148 |
| $B_{7,2}$ | 222975D325A487FE560A6D146311578D9C5501D28BC0A1FB48C9BDA173E869133A3<br>AA9506C42AE9F466E85611FC5F8F74E439638D66D2F00C682987A96D8887C |
| $B_{8,1}$ | 14B5F98E8D55FC8E9B4EE453C6963E052147A857AC1E08675D99A308E7269FAC560<br>0D7B155DE8CB1BAC786F45B46B523073692DE745FDF10724DDA38FD093B1C |
| $B_{8,2}$ | 1B71AFFB8117BCF8B5D002A99FEEA49503C0359B056963FE5271140E626F6F8FCE9<br>F29B37047F9CA89EBCE760405C6277F329065DF21AB3B779AB3E8C8955400 |
| $B_{9,1}$ | 0008B4E899E5F7E692BDCE69CE3FAD997183CFAEB2785D0C3D9CAE510316D4BD6<br>5A2A06CBA7F4E4C4A80839ACA81012343648EEA8DBBA2464A68E115AB3F4034 |
| $B_{9,2}$ | 5B7FE6808A10EA42FEF0ED9B41920F82023085C106FBBC1F56B567A14257021BC5F<br>DA60CBA05B08FAD6DC3B0410295884C7CCDE0E56347D649DE6DDCEEB0C95E |
| $B_{10,1}$ | 5E9B2B33EF82D0E64AA2226D6A0ADCD179D5932EE1CF401B336449D0FF775754CA<br>56650716E61A43F963D59865C7F017F53830514306649822CAA72C152F6EB2 |
| $B_{10,2}$ | 2CD8140C8A37DE0D0261259F63AA2A420A8F81FECB661DBA5C62DF6C817B4A61D<br>2BC1F068A50DFD0EA8FE1BD387601062E2276A4987A19A70B460C54F215E184 |
| $B_{11,1}$ | 06F1FF249192F2EAF063488E267EEE994E7760995C4FA6FFA0E4241825A7F5B65C74<br>FB16AC4C891BC008D33AD4FF97523EE5BD14126916E0502FF2F8E4A07FC2 |
| $B_{11,2}$ | 65287840D00243278F41CE1156D1868F24E02F91D3A1886ACE906CE741662B40B4EF |

| | |
|---|---|
| | DFB90F76C1ADD884D920AFA8B3427EEB84A759FA02E00635743F50B942F0 |
| $B_{12,1}$ | 4109DA2A24E41B1F375645229981D4B7E88C36A12DAB64E91C764CC43CCEC188EC 8C5855C8FF488BB91003602BEF43DBEC4A621048906A2CDC5DBD4103431DB8 |
| $B_{12,2}$ | 2185E3BC7076BA51AAD6B199C8C60BCD70E8245B874927136E6D8DD527DF0693D C10A1C8E51B5BE93FF7538FA138B335738F4315361ABF8C73BF40593AE22BE4 |
| $B_{13,1}$ | 228845775A262505B47288E065B23B4A6D78AFBDDB2356B392C692EF56A35AB4AA 27767DE72F058C6484457C95A8CCDD0EF225ABA56B7657B7F0E947DC17F972 |
| $B_{13,2}$ | 2630C6F79878E50CF5ABD353A6ED80BEACC7169179EA57435E44411BC7D566136D FA983019F3443DE8E4C60940BC4E31DCEAD514D755AF95A622585D69572692 |
| $B_{14,1}$ | 7273E8342918E097B1C1F5FEF32A150AEF5E11184782B5BD5A1D8071E94578B0AC7 22D7BF49E8C78D391294371FFBA7B88FABF8CC03A62B940CE60D669DFB7B6 |
| $B_{14,2}$ | 087EA12042793307045B283D7305E93D8F74725034E77D25D3FF043ADC5F8B5B186D B70A968A816835EFB575952EAE7EA4E76DF0D5F097590E1A2A978025573E |

Note that the numbers in the second column represent the hexadecimal representation of the first row of each circulant. Since there are only 511 possible positions, the leftmost bit is padded with a zero to allow a 128 digit hexadecimal number. Table 2 cannot be as efficiently described as table 1 due to the fact that the generator circulants do not have a low density of 1's.

# 9. APPENDIX B – COMPLEXITY

The complexity of LDPC codes has been an area of research and discussion. For a field programmable gate array (FPGA) or application specific integrated circuit (ASIC) implementation, the encoder's complexity are dominated by two factors: 1. the total number of required logic gates and 2. the routing complexity. For the code presented in this Orange Book, the quasi-cylic property allows for the use of shift registers whose required number of logic gates is proportional to n-k [6] or 8176-7156= 1020 (unshortened). In regards to the routing complexity, there is currently no way to predict this figure and would depend on a number of factors such as the choice of the FPGA or ASIC, routing algorithm and the layout of the device.

The decoder's complexity is larger than the encoder's and even more difficult predict. The primary complexity factors (the total number of required logic gates and the routing complexity) are a function of the choice of BP decoding algorithm (there are many) as well as the architectural decisions (i.e. parallel or serial processing, number of bits of finite precision, fixed number of iterations or stopping rule, use of look up tables, etc.) These choices also determine the decoder's bit error rate (BER) performance.

For the development of the baselined (8176, 7156) code, an FPGA implementation was used to confirm the software simulations. A Xilinx 8000 Virtex-2 FPGA was used for the test. The device contained both the encoder and decoder. The decoder algorithm was a Scaled Min-Sum parallel BP decoder (SMSPD) described in [7]. The encoder algorithm was a shift register based encoder described in [6]. An architectural evaluation was performed prior to implementation to produce a quasi-optimal implementation based on routing, logic requirements and BER performance.

The FPGA had the following statistics: 1. encoder used 2,535 logic slices out of 46,592 available or 5.4% and 4 memory blocks out of 168 available or 2.4%; 2. decoder used 21,803 logic slices out of 46,592 or 46.8% and 137 memory blocks out of 168 or 81.5%. The number of logic slices is an aggregate measure of the number of logic gates required and the routing complexity while the memory blocks figure is the number of dedicated FPGA memory blocks used. It is clear from these statistics that the encoder is of much lower complexity than the decoder using only 5.4% of the logic slices resources while the decoder requires 46%.

Appendix C summarizes the test results.

# 10. APPENDIX C – FPGA TEST RESULTS

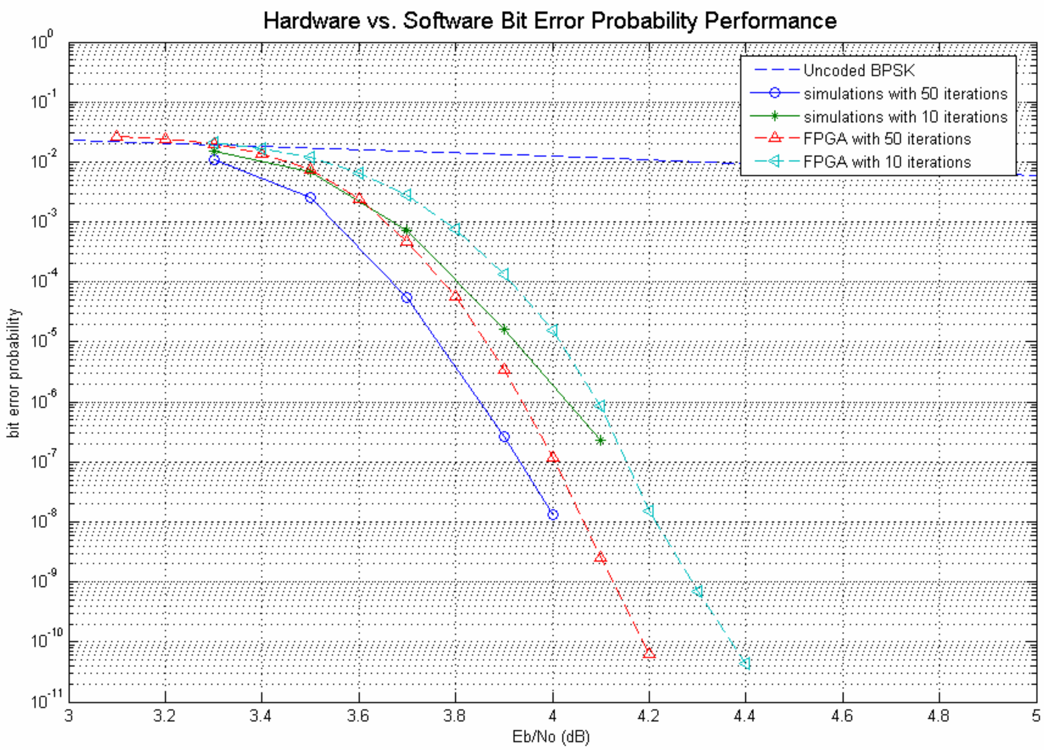


**Figure 7. Bit Error Rate Test Results**

Figure 7 shows the BER and Figure 8 shows the Block Error Rate (BLER) test results for 50 and 10 maximum iterations from an FPGA implementation of the baselined (8176, 7156) code. Note that for both cases the difference between simulations and hardware tests was 0.1 dB or less.

The encoder data rate was limited to 2 x system clock while the decoder operated at 14 x system clock / number of iterations. For testing, the system clock was set to 100 MHz, so for 10 iterations, the decoder operated at 140 Mbps. Although, the recommended shortened (8160, 7136) was not tested, it is reasonable to say that the baselined (8176, 7156) and the recommended shortened (8160, 7136) codes will have similar results.

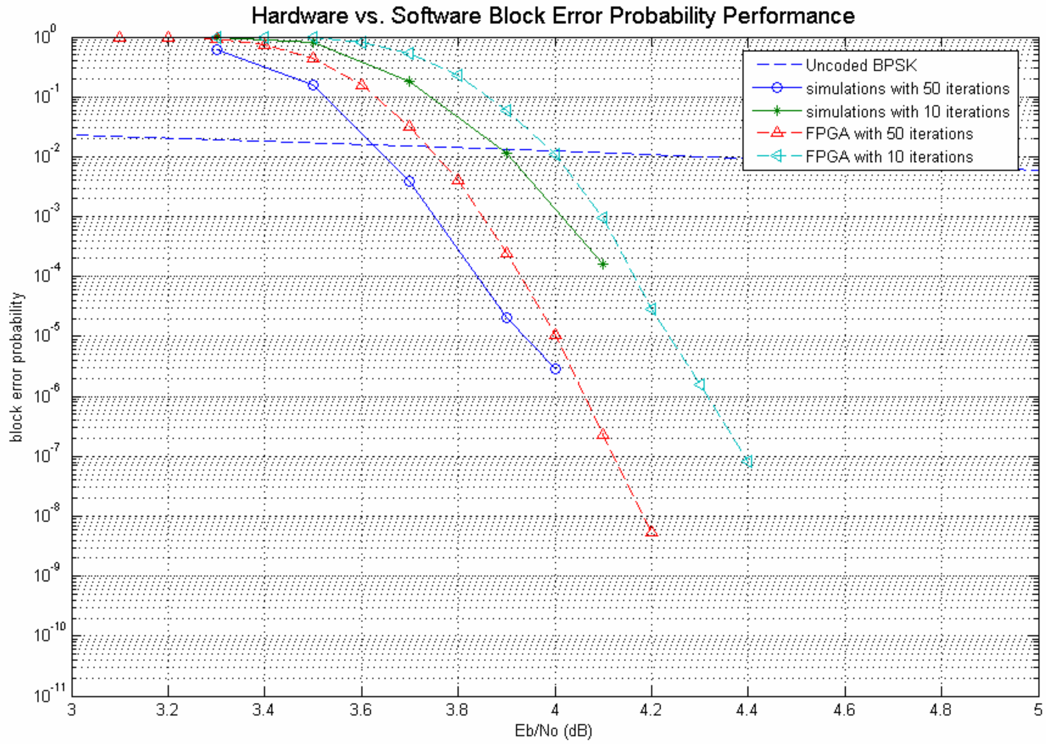


**Figure 8. Block Error Rate Test Results**